Offic de la propri´t´ intell ctu lle du Canada

Canadian Int ll ctual Property Offic

Un organisme d'Industrie Canada

An Agency of Industry Canada

OIPE JC139
OCT 0 1 2001
PATENT & TRADEMARK OFFICE

Bureau canadien des brevets

Certification

Canadian Patent Office

Certification

La présente atteste que les documents ci-joints, dont la liste figure ci-dessous, sont des copies authentiques des documents déposés au Bureau des brevets.

This is to certify that the documents attached hereto and identified below are true copies of the documents on file in the Patent Office.

Specification and Drawings, as originally filed, with Application for Patent Serial No: **2,310,150**, on May 30, 2000, by **CROSSKEYS SYSTEM CORPORATION,** assignee of Kelvin Edmison, Steve Bergwerff and Benoit Godin, for "Metadata-driven Statistics Translation".

RECEIVED
OCT 0 3 2001
Technology Center 2100

S. Grégoire

Agent certificateur/Certifying Officer

September 20, 2001

Date

Canada

OPIC CIPO

ABSTRACT OF THE DISCLOSURE

A method of converting data in one form to another isdisclosed, wherein the conversion is performed with metadata describing the original and final data.

Metadata-driven Statistics Translation

This invention relates to a method of converting data from one format to another.

Metadata is defined as data about data. Designing software to use metadata allows the software to adjust its behaviour based on the metadata. If the metadata is changed, then the behaviour of the software is correspondingly changed. This is the typical use of metadata.

Resolve is a performance management solution. Resolve must accept the various vendors' statistics files and normalize these statistics (by converting the format and meaning of statistics from a variety of vendors) to the standards-based set used within Resolve.

Typically this conversion is done using explicit programming.

According to the present invention there is provided a method of converting data in one form to another, wherein the conversion is performed with metadata describing the original and final data.

The conversion is preferably done through the use of three related sets of metadata. The first set of metadata describes the format and meaning of the statistics that the vendor supplies in the vendor statistics file. The second set of metadata describes the format and meaning of the Resolve statistics (commonly called Resolve Core Statistics) stored in the Resolve database. The third set of metadata describes the 'mappings' between the Resolve Core Statistics and the vendor statistics.

The use of this Metadata-driven statistics translation allows Resolve to accomplish the following:

- Remove the need to write programming code when supporting new statistics from existing vendors
- Significantly reduce the programming code required to support statistics from new vendors.

The vendor statistics metadata stores these attributes:

- A unique identifier

- Name
- A short description
- Placeholders for the vendor's unique identifier for the statistic

The Core Statistics metadata describes the following things about the Core Statistics

- A unique identifier
- Name
- A short description
- Database column name
- Order of column in database table
- Value type (e.g. integer, float, etc.)

The metadata describing the statistics mappings between the core statistics and the vendor statistics allows several vendor statistics to be added together, subtracted, multiplied or divided to form a core statistic. This statistics mapping also has a priority value.

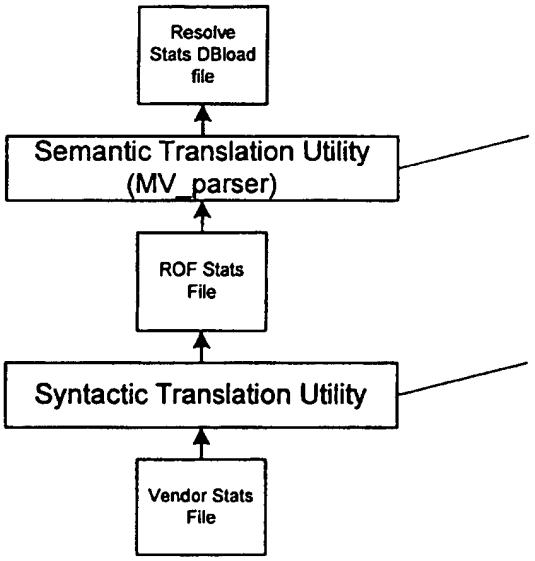
Each statistics mapping must be fully satisfied before it can be considered as a candidate for substitution as a core statistic. Since there may be multiple statistics mappings associated with a single core statistic, the metadata also permits the use of a priority value to assist the translation utility in determining which statistic mapping to use when multiple statistics mappings are fully satisfied.

The statistics metadata is used by two separate processes for two distinct purposes.

The Syntactic Translation Utility uses the metadata to determine which vendor statistics are of interest to Resolve (so that they can be extracted and passed on for processing).

The Semantic Translation Utility uses the metadata to correlate the vendor statistics as described in the ROF file with the statistics mappings. This utility also uses the statistics metadata to determine which fully satisfied mapping is used to map to the Core Statistic (according to the mapping's priority value).

The invention is illustrated by way of example in the accompanying drawings and described in more detail in the attached appendix.



The Semantic Translation Utility is used to convert the ROF statistics file from ROF format to the native format used to load into Resolve. The Semantic Translation Utility determines which type of native file is to be created, and thus can look up which vendor statistics are associated to it.

The Syntactic Translation Utility is used to convert the Vendor's statistics file from the native format to the ROF file format. The Attribute names used in the ROF file are looked up in the Vendor Statistics metadata by using the Vendor's statistics identification.

Marie Clerk

APPENDIX

LIST OF FIGURES

Figure 2-1 The Metadata Engine in the NI Architecture 4
Figure 3-1 Packages Classification 8
Figure 3-2 View of the Resolve Utilities Package 9
Figure 3-3 RS_ESQLC Access Module 10
Figure 3-4 View of the Metadata_Cache Package 11
Figure 3-5 View of the Syntactic Utilities Package 24
Figure 3-6 View of the Semantic Utilities Package 26
Figure 4-1 Resolve Statistics Cache - Creation and Configuration 29
Figure 4-2 Resolve Statistics Cache - Loading 31
Figure 4-3 SyntactiCacheMgr interacts with Syntactic Translator 33
Figure 4-4 Semantic Translator interacts with Semantic Translation Utility Objects ... 35

1 INTRODUCTION

1.1 Purpose

This document provides the object-oriented design of a suite of library functions to support the Resolve stats importor in a multi-vendor metadata-driven framework. In this document, such a suite of library functions are called Metadata Engine.

1.2 Organization

This document is organized in the following order:

- Section 1 – *Introduction* – Introduction to the document and the feature
- Section 2 – *Design Overview* – Provides design motivations such as why this feature is being develop, discuss the external dependencies affecting the current design of the feature as well as the architectural goals and constraints.
- Section 3 – *Logical Architecture* – Provides a detailed description of each of the packages and their contents.

- Section 4 – *Interaction Diagrams* – Provides scenario diagram that will help understanding how the syntactic and semantic translators interact with the Metadata Engine.

2 DESIGN OVERVIEW

2.1 Motivations

The multi-vendor features (MV) from Resolve Dynamite release provides a framework to build the network interfaces (NIs) for new vendor equipment and/or new management system. In this MV framework, stats importing and MV Parser, as one of the tasks carried out by the NI, will be based upon the intelligence provided by the Metadata instead of on the vendor-specific knowledge hard coded in the application program. The Metadata makes its intelligence available to the stats importer and MV Parser through a collection of class objects in C++ which is named as Metadata Engine in this document.

Briefly, Metadata Engine will answer following questions from the SI, MV Parser and other clients:

1. Is the NMS stats needed for the Resolve stats reporter?
2. Is the NMS stats a supplemental one?
3. What are those NMS stats needed to calculate a specific Resolve stats?
4. What are those Resolve stats and their positions needed for a dbload record?

2.2 System Overview

The Metadata Engine for Dynamite release is designed to support the MV stats importer and parser. Therefore it is an extension of the Resolve Network Interface.

Figure 2-1 shows the interaction among various components of the Resolve Network interface.

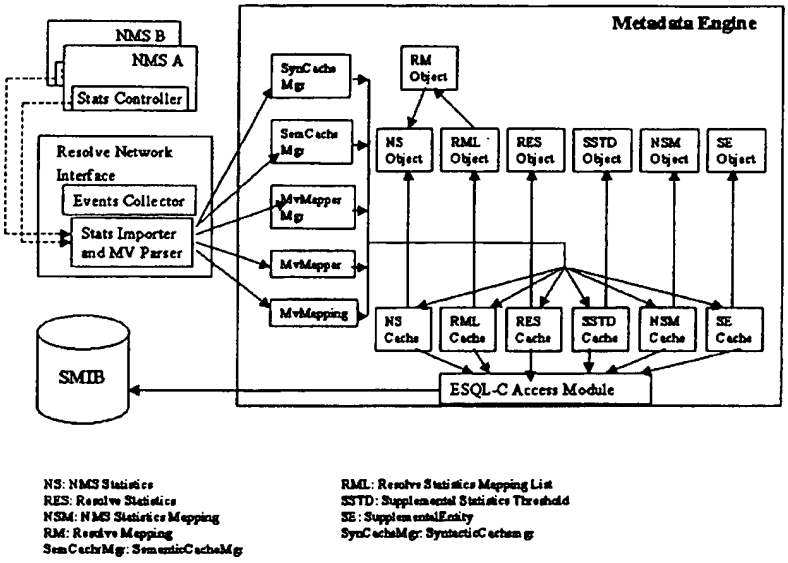


Figure 2-1 The Metadata Engine in the NI Architecture

As shown in the figure above, the Statistics Importer and MV Parser processes interact with the various Metadata cache manager objects to query various NMS and Resolve Metadata objects. These cache manager objects themselves interact with various cache objects containing all the Metadata information.

The Stats Importer and MV Parser processes will ensure these caches are properly configured and loaded before they can be used. This is done by invoking several member methods of Syntactic and Semantic Cache Manager class.

Different Metadata caches are to be constructed for the syntactic and semantic translation. For syntactic translation, four Metadata caches are needed (see Table 2-1). For semantic translation, four different Metadata caches are needed (see Table 2-2).

All these caches are constructed from six Metadata tables stored in the database, they are

- NMS stats Metadata table (i.e., vendorstatdesc),
- Resolve stats Metadata table (i.e., tdz_statdesc),
- Resolve stats mapping table (i.e., statmap),
- Supplemental stats threshold description table (i.e., suppstatthresh),
- resource table (i.e., ckresouce), and
- MV translation table (i.e., tmx_mvxlate).

Cache Name	Functionality	Description
NMS Statistics Cache	speedy access to the attributes of NMS Metadata	all NMS stats Metadata for a specific type of NMS (e.g., for Newbridge 46020 version 2.0). For details see section 3.3.5.3.3.
NMS Statistics Mapping Cache	speedy checking to see if the NMS stats is related to a Resolve stats	all NMS Metadata (of a specific type of NMS) having associated Resolve stats
Supplemental Entity Cache	speedy checking to see if all stats of an entity have been selected as supplemental stats	all entity (of a specific NMS type) with their supplemental flag set to be on.
Supplemental Statistics Cache	speedy checking to see the specific stat is a supplemental one	all Supplemental for a specific type of NMS.

Table 2-1 Metadata caches to be constructed for syntactic translation

Cache Name	Functionality	Description
Resolve Statistics Cache	speedy access to all attributes of Resolve Metadata	all Resolve Metadata for a specific type of NMS and a specific type of technology
Resolve Statistics Mapping Cache	speedy access to the list of NMS stats associated to the Resolve stats	all Resolve stats and their mappings for the specified NMS type and specified technology
Resolve Statistics Mapping List Cache	speedy access to a list of Resolve Statistics Mappings for a Resolve stats	all Resolve stats and their mappings for the specified NMS type and specified technology
NMS Statistics cache	speed access to all attributes of NMS stats	all NMS Metadata as loaded in the Resolve Statistics Mapping Cache.

Table 2-2 Metadata caches to be constructed for semantic translation

2.3 Architectural Goals and Constraints

The main architecture goals are:

1. fast query of Metadata information to support syntactic and semantic translation of stats from its native NMS format into corresponding stats in the format defined by Resolve Metadata.
2. flexible design architecture to support future functionality development of Metadata Engine.

The first goal is addressed by loading the necessary Metadata information into process's memory caches and careful selection of lookup method.

The second goal is addressed by modelling NMS stats Metadata and Resolve stats Metadata as instances of class in C++ with all columns in the Metadata tables as attributes of the class. This makes it relatively easy to provide further intelligence from the Metadata to meet the requirement from various applications.

There are basically two approaches to import those tables as listed in the previous section in order to construct and load the Metadata caches:

1. Using ESQL to invoke Informix APIs
2. Building a Generic DBA Client to communicate with Resolve DBA Server

Both approaches have its advantages and disadvantages (Table 2-3). Due to the time constraints, we will use existing ESQL database access module to access the database for this release. A DBA client class will be developed in the future release.

	Advatanges	Disadvantages
ESQL	better performance and less development time	not consistent with object-oriented methodology as used by rest of the design.
DBA Client	• unified object-oriented methodology through the design • Metadata schema change would break the client code	potential performance bottleneck when communicating with DBA Server. More development time required.

Table 2-3 Two approaches to access the Metadata database

3 LOGICAL ARCHITECTURE

3.1 Architecture Overview

The Metadata Engine will require major development in the following areas:

- Caches – will be implemented using hash tables.
- Several manager objects – will be implemented as C++ classes.

The Metadata Engine will make use of following existing utilities to meet its database access, configuration and message logging requirements:

- ESQL Access Module
- Configuration utility (CK_Config)
- Logging utility (CK_Log)

3.2 Package Classification

The system is broken down into 7 packages and grouped into 2 collections of categories (see Figure 3-1). The top and upper middle four categories are packages that are not specific to the Metadata Engine. The lower midlle three categories are packages and classes specific to the Metadata Engine. The bottom two client packages represent the Statistics Importer and MV Parser application processes which are going to interact directly with the syntactic and semantic utilities categories, respectively.

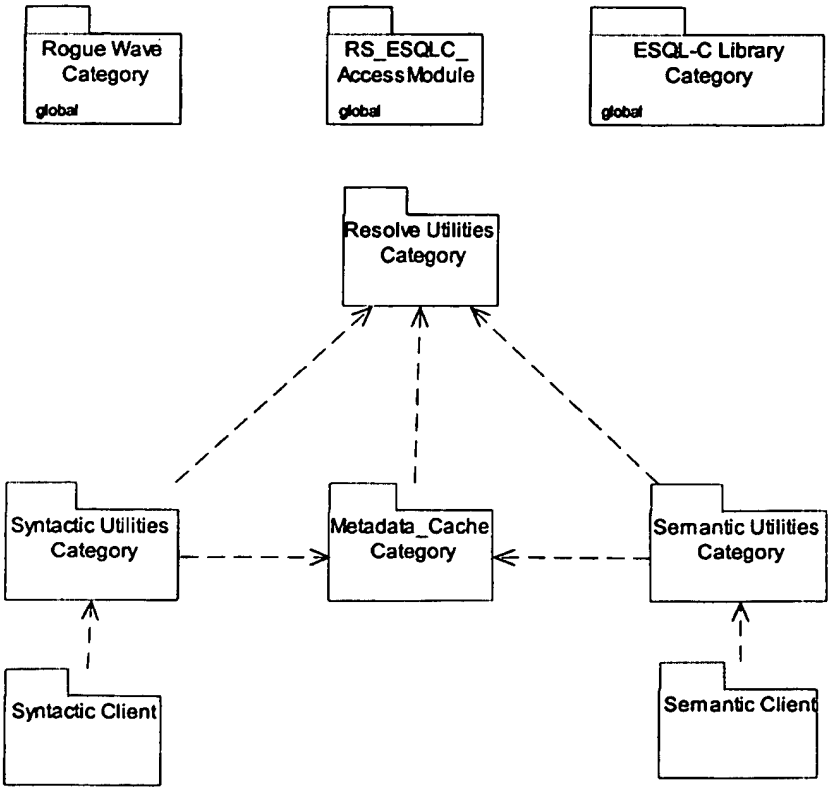


Figure 3-1 Packages Classification

3.3 Package and Classes – Detailed Description

3.3.1 Package – Rogue Wave

3.3.1.1 Rogue Wave – Package Description

Rogue Wave is a third party software that consist of libraries of generic classes such as list and hash table. Rogue Wave is widely used throughout the Resolve product.

3.3.1.2 Rogue Wave – Package Contents

Rogue Wave classes that will be of most interest to us are:

- RWHashTable –a simple hash table class to store type of objects.
- RWSlistCollectables – a class to represent a group of ordered RWCollectable type of elements, without keyed access.
- RWCollectable – this type of objects enable its user to store it in a collection (hash table, list, etc.)

3.3.2 Package – Resolve Utilities

3.3.2.1 Resolve Utilities – Description

The Resolve Utilities package contains classes that are generic enough so that they can be used by many sub-systems of Resolve. These utilities include some existing classes such as CK_Config and CK_Log which are used for configuration management and message logging. Other utilities included in this package are being built to support Resolve's Naming Service Engine (for detailed design information see [4])

3.3.2.2 Resolve Utilities – Contents

Figure 3-2 shows those classes making up this package. Note that only the ones used by the Metadata Engine are shown. Since many of the classes shown in Figure 3-2 have been documented in the Naming Service Engine OOD, only essential summarization on these classes are provided in this section to avoid unnecessary duplication.

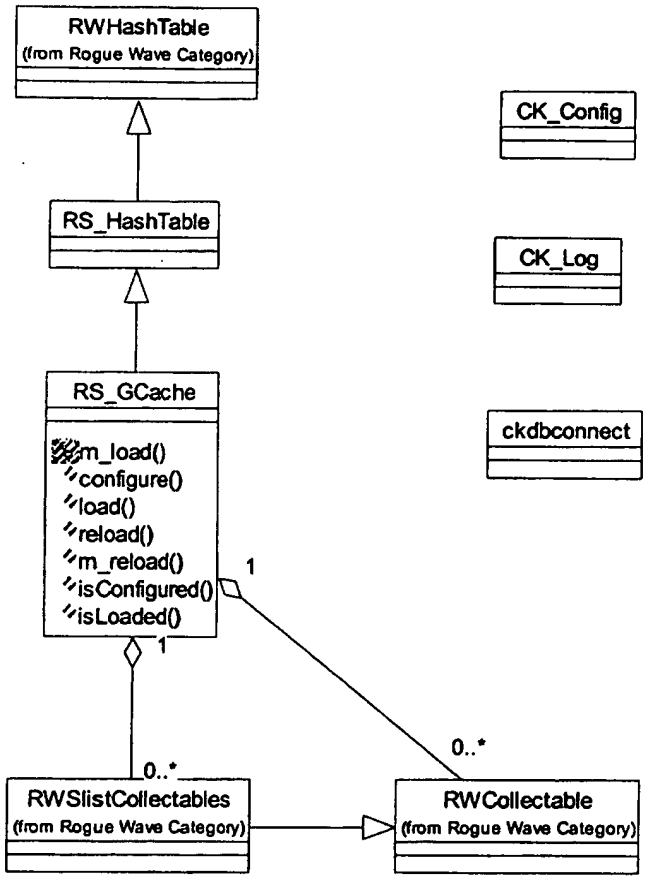


Figure 3-2 View of the Resolve Utilities Package

3.3.2.3 Resolve Utilities – Classes

3.3.2.4 Class RS_HashTable

Class name: RS_HashTable

Superclass: RWHashTable

Class role: This class offers the exact same capability as provided by its superclass RWHashTable except that it adds one additional functionality capable of automatically resizing the size of the hash table. This functionality can be used to prevent excessive linear search in the hash table.

3.3.2.5 Class RS_GCache

Class name: RS_GCache

Superclass: RS_HashTable

Class role: This class is an abstract class for hash table type of caches. This class is in some way a template for its child cache classes. Each child class will have to adopt certain behaviors to specify its own *hash* and *isEqual* functions.

3.3.3 Package – ESQL-C Library (API)

ESQL-C is an SQL API that enables the user to connect to database and embedded Structured Query Language statements into a C program. For more information about ESQL-C consult the *Informix ESQL-C Programmer's Manual.*

3.3.4 Package – RS_ESQLC Access Module

This package consists of a C wrapper around the ESLQ-C API. This wrapper's main responsibility is to create an abstraction layer between the ESQL-C API and the rest of the Metadata Engine (see Figure 3-3).

This package consists of a suite of C functions. Detailed information about these functions can be found in the Naming Service Engine OOD document.

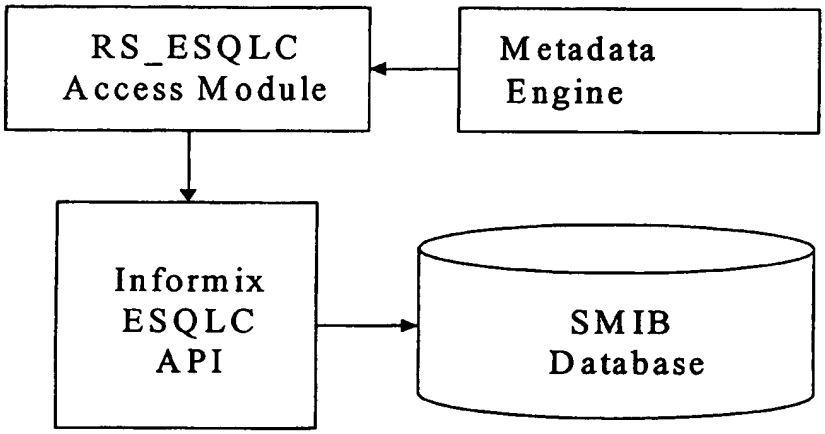


Figure 3-3 RS_ESQL C Access Module

3.3.5 Package – Metadata_Cache

3.3.5.1 Metadata_Cache – Description

This package contains the classes that make up the caches required by the Metadata Engine. Six caches will be required:

1. *MD_NMSStatCache* – for NMS stats metadata lookup.
2. *MD_ResStatCache* – for Resolve stats metadata lookup
3. *MD_ResStatMappingCache* – for NMS stats to Resolve stats mapping lookup
4. *MD_NMSStatMappingCache* – for NMS stats to technology type mapping lookup
5. *MD_SupplementalCache* – for NMS supplemental stats threshold mapping lookup
6. *MD_SupplementalEntityCache* – for NMS supplemental entity lookup.

3.3.5.2 Metadata_Cache – Contents

Figure 3-4 shows which components are involved in this package, and how they are related together.

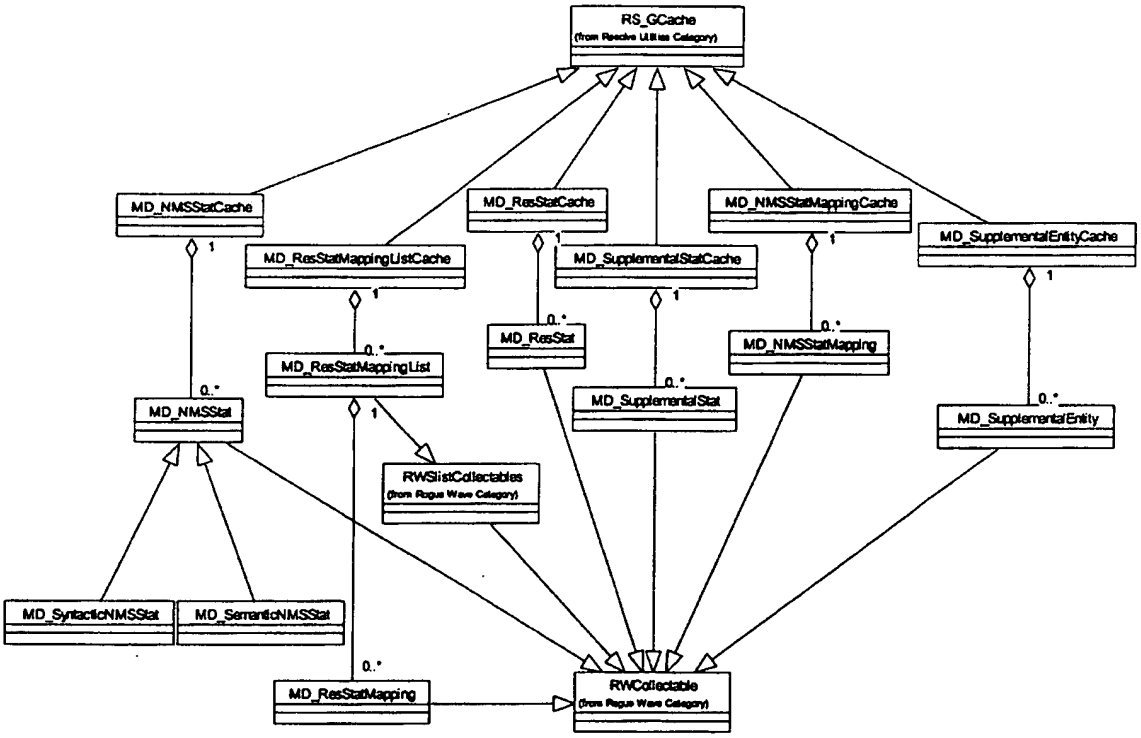


Figure 3-4 View of the Metadata_Cache Package

The package contains the following new classes:

- MD_SyntacticNMSStat
- MD_SemanticNMSStat
- MD_NMSStat
- MD_NMSStatCache
- MD_ResStatMapping
- MD_ResStatMappingList
- MD_ResStatMappingListCache
- MD_ResStat
- MD_ResStatCache

- MD_SupplementalStat
- MD_SupplementalStatCache
- MD_NMSStatMapping
- MD_NMSStatMappingCache
- MD_SupplementalEntity
- MD_SupplementalEntityCache

Details on each of the classes are given in the section that follows.

3.3.5.3 Metadata_Cache – Classes

3.3.5.3.1 Class MD_SyntacticNMSStat

Class name: MD_SyntacticNMSStat

Superclass: MD_NMSStat

Class role: This class represents a NMS stats object. It implements a *hash* function and an *isEqual* function suitable to the construction of class MD_NMSStatCache for fast lookup in the syntactic translation stage of stats importing.

Methods:

Name	Description	Return value
constructor	Constructs a MD_SyntacticNMSStat object.	
hash	Returns a hash value, so that the object can be stored in the RWHashTable. The hash value returned the NMS statid* or statstr**.	A hash value
isEqual	Method to call to compare this object with another one. Since lookup in the cache will be based on the NMS *statid* or *statstr*, two NMS stats will be considered equal if their *statids* or *statstrs* are identical.	TRUE or FALSE

Table 3-1 MD_SyntacticNMSStat Methods

(*) In case of NMS statid, if two ids are involved (as being the case for NB 46020), a simple hash value can be calculated using the following formula:
srcstatid1 * k + srcstatid2

where constant k is assigned with a big number (say, 32,000) to minimize the number of objects hashed into the same bucket.

(**) In case of statstr, the string can be converted into a numeric value using some simple algorithm.

3.3.5.3.2 Class MD_SemanticNMSStat

Class name: MD_SemanticNMSStat

Superclass: MD_NMSStat

Class role: This class represents a NMS stats object. It implements a *hash* function and an *isEqual* function suitable to the construction of class MD_NMSStatCache for fast lookup in the semantic translation stage of stats importing.

Methods:

Name	Description	Return value
constructor	Constructs a MD_SemanticNMSStat object.	
hash	Returns a hash value, so that the object can be stored in the RWHashTable. The hash value returned the de_type value of the NMS stat.	A hash value
isEqual	Method to call to compare this object with another one. Since lookup in the cache will be based on the *de_type* of the NMS stat, two NMS stats will be considered equal if their *de_types* are identical.	TRUE or FALSE

Table 3-2 MD_SemanticNMSStat Methods

3.3.5.3.3 Class MD_NMSStat

Class name: MD_NMSStat

Superclass: RWCollectable

Class role: This class represents a NMS stats object with following attributes

- de_type
- de_name
- de_description
- valuetype
- statunits
- statstrategy
- statperiod
- srcstatid1
- srcstatid2
- srcstatstr
- rcaname
- nmstype

This class will provide interface to get all the attributes required by the syntactic or semantic stage of translation. It does not implement its own hash and isEqual methods.

Methods:

Name	Description	Return value
constructor	Constructs a MD_NMSStat object.	
getDe_type	Returns stat's de_type	A short integer
getRcaname	Returns stat's rcaname	A RWCString string
getSrcstatid1	Returns the stat's srcstatid1	An int integer
getSrcstatid2	Returns the stat's srcstatid2	An int integer

Table 3-3 MD_NMSStat Methods

3.3.5.3.4 Class MD_NMSStatCache

Class name: MD_NMSStatCache

Superclass: RS_GCache

Class role: This class maintains a cache of either MD_SyntacticNMSStat or MD_SemanticNMSStat objects.

Methods:

Name	Description	Return value
constructor	Receives a pointer to a CK_Config and CK_Log object. Also accepts optional arguments for size and fill factor of the hash table cache.	
configure	Gets its configuration using the CK_Config object. If the configuration is successful sets its private member m_configured to true. Otherwise sets it to false. The configuration parameters that it will belooking for are DB_NAME and DB_SERVERNAME.	TRUE or FALSE
m_load	Check if the cache is properly configured. If it is it attempts to load the cache by querying the database using the RS_ESQLC Access Module. If successful sets its private member m_loaded to true otherwise sets it to false.	TRUE or FALSE
m_reload	Attempts to reload the cache using the m_load method.	TRUE or FALSE
getDe_type	Retrieves the de_type attribute of the object it	TRUE or FALSE

	maintains based on its statid. Useful for syntactic stage of translation.	
getSrcstatid	Retrieves the statid1 (and statid2 if applicable) attribute(s) of the object it maintains based on its de_type. Useful for semantic stage of translation.	TRUE or FALSE
getRcaname	Retrieves the rcaname attribute of the object it maintains based on its de_type. Useful for semantic stage of translation.	TRUE or FALSE
getRcaname	Retrieves the rcaname attribute of the object it maintains based on its statid. Useful for syntactic stage of translation.	TRUE or FALSE

Table 3-4 MD_NMSStatCache Methods

3.3.5.3.5 Class MD_ResStatMapping

Class name: MD_ResStatMapping

Superclass: RWCollectable

Class role: This class represents a MD_ResStatMapping object with following attributes:

- Resolve_de_type
- Nms_stat1_de_type
- Nms_stat2_de_type
- Nms_stat3_de_type
- Operation1
- Operation2

Methods:

Name	Description	Return value
constructor	Constructs a MD_ResStatMapping object.	
getDe_type	Retrieves the Resolve_de_type attribute of the object it maintains based on its Resolve_de_type.	A short integer
getNMSStat1	Retrieves the Nms_stat1_de_type attribute of the object it maintains based on its Resolve_de_type.	A short integer
getNMSStat2	Retrieves the Nms_stat2_de_type attribute of the object it maintains based on its Resolve_de_type.	A short integer

getNMSStat3	Retrieves the Nms_stat3_de_type attribute of the object it maintains based on its Resolve_de_type.	A short integer
getOperation1	Retrieves the operation1 attribute of the object it maintains based on its Resolve_de_type.	A RWCString string
getOperation2	Retrieves the operation2 attribute of the object it maintains based on its Resolve_de_type.	A RWCString string

Table 3-5 MD_ResStatMapping Methods

3.3.5.3.6 Class MD_ResStatMappingList

Class name: MD_ResStatMappingList

Superclass: RWSlistCollectables

Class role: This class represents a MD_ResStatMappingList objects with following attributes:

- Resolve_de_type
- mappingCounter

Methods:

Name	Description	Return value
constructor	Constructs a MD_ResStatMappingList object.	
hash	Returns a hash value, so that the object can be stored in the RWHashTable. The hash value returned should simply be the Resolve_de_type.	A hash value
isEqual	Method to call to compare this object with another one. Two Resolve mappingList objects will be considered equal if their *Resolve_de_types* are identical.	TRUE or FALSE
getNextMapping	Returns a MD_ResStatMapping object indexed by its mappingCounter.	A MD_ResStatMapping object.

Table 3-6 MD_ResStatMappingList Methods

3.3.5.3.7 Class MD_ResStatMappingListCache

Class name: MD_ResStatMappingListCache

Superclass: RS_GCache

Class role: This class maintains a cache containing a list of ResStatMapping objects for each applicable Resolve stat.

Methods:

Name	Description	Return value
constructor	Receives a pointer to a CK_Config and CK_Log object. Also accepts optional arguments for size and fill factor of the hash table cache.	
configure	Gets its configuration using the CK_Config object. If the configuration is successful sets its private member m_configured to true. Otherwise sets it to false. The configuration parameters that it will belooking for are DB_NAME and DB_SERVERNAME.	TRUE or FALSE
m_load	Check if the cache is properly configured. If it is it attempts to load the cache by querying the database using the RS_ESQLC Access Module. If successful sets its private member m_loaded to true otherwise sets it to false.	TRUE or FALSE
m_reload	Attempts to reload the cache using the m_load method.	TRUE or FALSE
getNumResStatMapping	Retrieves the number of Resolve stats mapping objects maintained by the mapping list based on its Resolve_de_type.	TRUE or FALSE

Table 3-7 MD_ResStatMappingListCache Methods

3.3.5.3.8 Class MD_ResStat

Class name: MD_ResStat

Superclass: RWCollectable

Class role: Each instance of this class represents a Resolve stats metadata object with following attributes:

- de_type
- de_name
- de_description
- dz_entitydescription
- valuetype
- statunits
- statcolumn

- statiscore
- statstrategy
- columnorder

Methods:

Name	Description	Return value
constructor	Constructs a MD_ResStat object	
hash	Returns a hash value, so that the object can be stored in the RWHashTable. The hash value returned should simply be the columnorder.	A hash value
isEqual	Method to call to compare this object with another one. Two Resolve stats (of the same NMS type and techtype) objects will be considered equal if their *columnorders* are identical.	TRUE or FALSE
getDe_type	Retrieves the de_type attribute of the object it maintains based on its columnorder.	A short integer

Table 3-8 MD_ResStat Methods

3.3.5.3.9 Class MD_ResStatCache

Class name: MD_ResStatCache

Superclass: RS_GCache

Class role: This class maintains a cache for MD_ResStat objects.

Methods:

Name	Description	Return value
constructor	Receives a pointer to a CK_Config and CK_Log object. Also accepts optional arguments for size and fill factor of the hash table cache.	
configure	Gets its configuration using the CK_Config object. If the configuration is successful sets its private member m_configured to true. Otherwise sets it to false. The configuration parameters that it will belooking for are DB_NAME and DB_SERVERNAME.	TRUE or FALSE
m_load	Check if the cache is properly configured. If it is it attempts to load the cache by querying the database using the RS_ESQLC Access Module. If successful sets its private member m_loaded to true otherwise	TRUE or FALSE

	sets it to false.	
m_reload	Attempts to reload the cache using the m_load method.	TRUE or FALSE
getDe_type	Retrieves the de_type of the Resolve stats based on its columnorder	TRUE or FALSE

Table 3-9 MD_ResStatCache Methods

3.3.5.3.10 Class MD_SupplementalStat

Class name: MD_SupplementalStat

Superclass: RWCollectable

Class role: Each instance of this class represents a NMS supplemental stats object with following attributes:

- srcstatid1
- srcstatid2
- srcstatstr
- NMS_de_type
- MinValue
- MaxValue

Methods:

Name	Description	Return value
constructor	Constructs a supplementalStat object	
hash	Returns a hash value, so that the object can be stored in the RWHashTable. The hash value returned the NMS statid or statstr.	A hash value
isEqual	Method to call to compare this object with another one. Two NMS stats objects will be considered equal if their *srcstatid1*s and *srcstatid2*s or their *srcstatstr*s are identical.	TRUE or FALSE
getDe_type	Retrieves the de_type of the supplemental stats based on its statid or statstr	a short integer
getMaxvalue	Retrieves the maxValue of the supplemental stats based on its statid or statstr	a numeric value
getMinvalue	Retrieves the minValue of the supplemental stats based on its statid or statstr	a numeric value

Table 3-10 MD_SupplementalStat Methods

3.3.5.3.11 Class MD_SupplementalStatCache

Class name: MD_SupplementalStatCache

Superclass: RS_GCache

Class role: This class maintains a cache for MD_SupplementalStat objects.

Methods:

Name	Description	Return value
constructor	Receives a pointer to a CK_Config and CK_Log object. Also accepts optional arguments for size and fill factor of the hash table cache.	
configure	Gets its configuration using the CK_Config object. If the configuration is successful sets its private member m_configured to true. Otherwise sets it to false. The configuration parameters that it will belooking for are DB_NAME and DB_SERVERNAME.	TRUE or FALSE
m_load	Check if the cache is properly configured. If it is it attempts to load the cache by querying the database using the RS_ESQLC Access Module. If successful sets its private member m_loaded to true otherwise sets it to false.	TRUE or FALSE
m_reload	Attempts to reload the cache using the m_load method.	TRUE or FALSE
getDe_type	Retrieves the de_type of the supplemental stat based on its statid or statstr	TRUE or FALSE
getMaxvalue	Retrieves the maxValue of the supplemental stats based on its statid or statstr	TRUE or FALSE
getMinvalue	Retrieves the minValue of the supplemental stats based on its statid or statstr	TRUE or FALSE

Table 3-11 MD_ SupplementalStatCache Methods

3.3.5.3.12 Class MD_NMSStatMapping

Class name: MD_NMSStatMapping

Superclass: RWCollectable

Class role: Each instance of this class represents a NMS stats to techtype mapping object with following attributes

- srcstatid1

- srcstatid2
- srcstatstr
- NMS_de_type
- Techtype
- techtype_key

Methods:

Name	Description	Return value
constructor	constructs a MD_NMSStatMapping object	
hash	Returns a hash value, so that the object can be stored in the RWHashTable. The hash value returned the NMS statid or statstr.	A hash value
isEqual	Method to call to compare this object with another one. Two NMS stats mapping objects will be considered equal if their *srcstatid1*s and *srcstatid2*s or their *srcstatstrs* are identical.	TRUE or FALSE
getDe_type	Retrieves the NMS_de_type of the NMSStatMapping object based on its statid or statstr	a short integer
getTechtype	Retrieves the techtype of the NMSStatMapping object based on its statid or statstr	a short integer
getKey	Retrieves the techtype_key of the NMSStatMapping object based on its statid or statstr	an integer

Table 3-12 MD_NMSStatMapping Methods

3.3.5.3.13 Class MD_NMSStatMappingCache

Class name: MD_NMSStatMappingCache

Superclass: RS_GCache

Class role: This class maintains a cache for MD_NMSStatMapping objects.

Methods:

Name	Description	Return value
constructor	Receives a pointer to a CK_Config and CK_Log object. Also accepts optional arguments for size and fill factor of the hash table cache.	
configure	Gets its configuration using the CK_Config object. If the configuration is successful sets its	TRUE or FALSE

	private member m_configured to true. Otherwise sets it to false. The configuration parameters that it will belooking for are DB_NAME and DB_SERVERNAME.	
m_load	Check if the cache is properly configured. If it is it attempts to load the cache by querying the database using the RS_ESQLC Access Module. If successful sets its private member m_loaded to true otherwise sets it to false.	TRUE or FALSE
m_reload	Attempts to reload the cache using the m_load method.	TRUE or FALSE
getDe_type	Retrieves the NMS_de_type of the NMSStatMapping object based on its statid or statstr	TRUE or FALSE
getTechtype	Retrieves the techtype of the NMSStatMapping object based on its statid or statstr	TRUE or FALSE
getKey	Retrieves the techtype_key of the NMSStatMapping object based on its statid or statstr	TRUE or FALSE

Table 3-13 MD_NMSStatMappingCache Methods

3.3.5.3.14 Class MD_SupplementEntity

Class name: MD_SupplementalEntity

Superclass: RWCollectable

Class role: Each instance of this class represents an entity object whose stats all have been selected as being supplemental. There is one attribute for the entity object:

- entitystr

Methods:

Name	Description	Return value
constructor	constructs a MD_supplementEntity object	
hash	Returns a hash value, so that the object can be stored in the RWHashTable. The hash value returned a numeric value converted from the entitystr of the object.	A hash value
isEqual	Method to call to compare this object with another one. Two supplemental entity objects will be considered equal if their *statstr*s are identical.	TRUE or FALSE

Table 3-14 MD_SupplementalEntity Methods

3.3.5.3.15 Class MD_SupplementalEntityCache

Class name: MD_SupplementalEntityCache

Superclass: RS_GCache

Class role: This class maintains a cache for MD_SupplementalEntity objects.

Methods:

Name	Description	Return value
constructor	Receives a pointer to a CK_Config and CK_Log object. Also accepts optional arguments for size and fill factor of the hash table cache.	
configure	Gets its configuration using the CK_Config object. If the configuration is successful sets its private member m_configured to true. Otherwise sets it to false. The configuration parameters that it will belooking for are DB_NAME and DB_SERVERNAME.	TRUE or FALSE
m_load	Check if the cache is properly configured. If it is it attempts to load the cache by querying the database using the RS_ESQLC Access Module. If successful sets its private member m_loaded to true otherwise sets it to false.	TRUE or FALSE
m_reload	Attempts to reload the cache using the m_load method.	TRUE or FALSE

Table 3-15 MD_SupplementalEntityCache methods

3.3.6 Package – Syntactic Utilities

3.3.6.1 Syntactic Utilities – Description

This package contains the classes that the user (the syntactic translator process) of the Metadata Engine will have to deal with when convert NMS stats into ROF files.

3.3.6.2 Syntactic Utilities – Contents

As shown in the figure below, this package contains one new class:

- *MD_SyntacticCacheMgr* – a class to load the syntactic metadata caches and to check to see if the NMS stats being process is required to generate final dbload file

The use of the object of the class above in conjunctions with the required valid caches will help the syntactic translator to filter out those NMS stats which are not applicable for the ROF file associated with the specific type of NMS and specific type of technology. It also help identify those NMS stats which will be collected as supplemental stats.

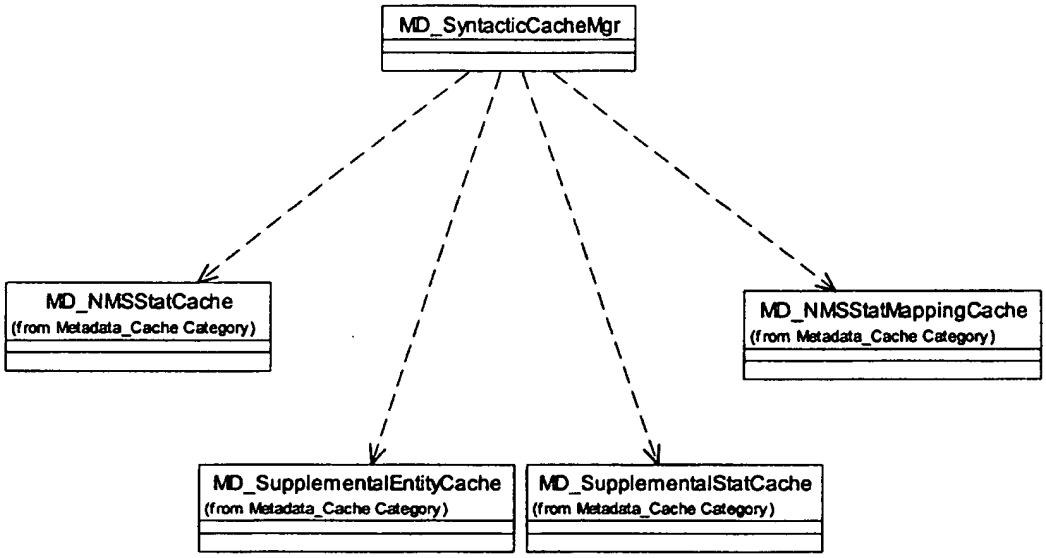


Figure 3-5 View of the Syntactic Utilities Package

3.3.6.3 Class

3.3.6.3.1 Class MD_SyntacticCacheMgr

Class name: MD_SyntacticCacheMgr

Superclass: none

Class role: This class is responsible for loading the required metadata caches and determines if a NMS stats is required and which techtype it is associated to.

Methods:

Name	Description	Return value
constructor	construct a MD_SyntacticCacheMgr object	
mvSyntacticInit	construct and load a cache for all NMS Stats	TRUE or FALSE

	belonging to NMS_type	
mvRegister	construct and load a cache for mapping NMS stats to corresponding Resolve stat	TRUE or FALSE
mvRegisterSupplementalStat	construct and load a cache for supplemental stats with threshold value defined and a cahe for all entity with all its stats selected as supplemental stats.	TRUE or FALSE
isNMSStatRequired	check to see if the NMS stats is a core or extended one?	An integer
isASupplementalStat	check to see if the NMS stats is a supplemental one?	An integer

Table 3-16 MD_SyntacticCacheMgr Methods

3.3.7 Package – Semantic Utilities

3.3.7.1 Semantic Utilities – Description

This package contains the classes that the user (the semantic translator process) of the Metadata Engine will have to deal with to convert NMS stats from ROF files into dbload files.

3.3.7.2 Semantic Utilities – Contents

As shown in the figure below, this package contains four new classes:

- *MD_SemanticCacheMgr* – a class to construct and load the semantic metadata caches
- *MD_MvStatMapperMgr* – a managing class to control the mappings of NMS stats to all Resolve statictics for one NMS_type and one techtype.
- *MD_MvStatMapper* – a class containing the mapping(s) of several NMS stats to one Resolve stats.
- *MD_MvStatMapping* – a class containing one mapping of several NMS stats to one Resolve stats and a pointer pointing to the cache for all NMS stats.

The use of the objects of the classes above in conjunctions with the required valid caches will help the semantic translator to calculate each of the Resolve stats for the dbload file.

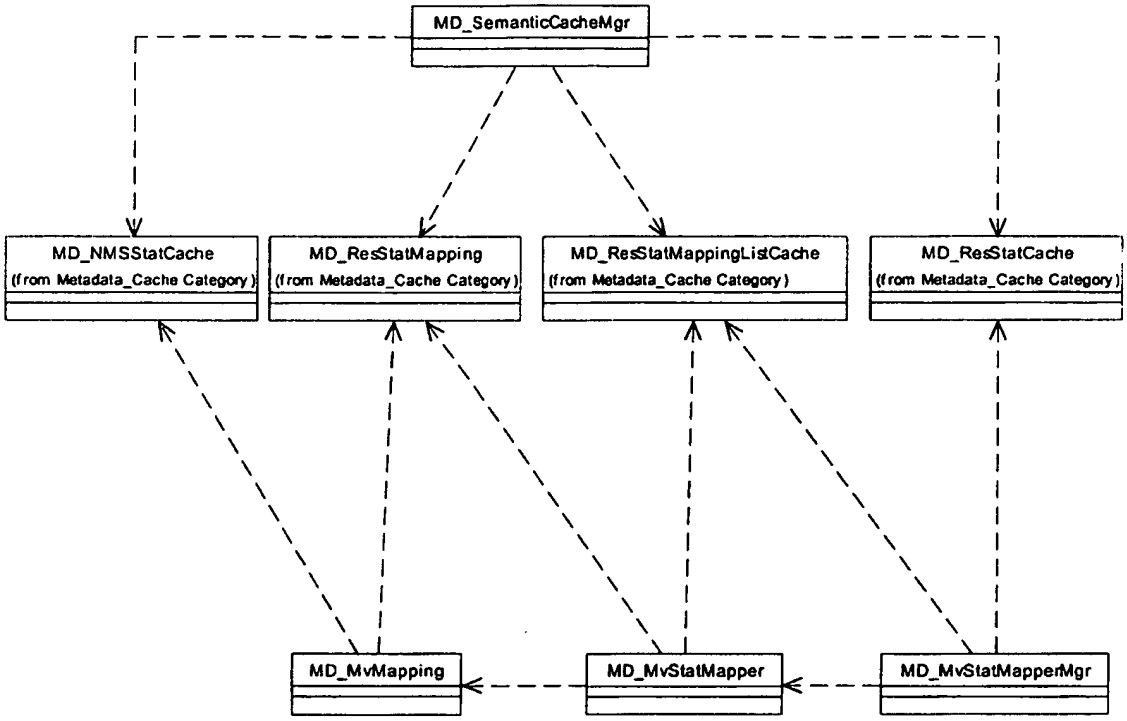


Figure 3-6 View of the Semantic Utilities Package

3.3.7.3 Classes

3.3.7.3.1 Class MD_SemanticCacheMgr

Class name: MD_SemanticCacheMgr

Superclass: none

Class role: This class is responsible for loading the required metadata caches

Methods:

Name	Description	Return value
constructor	constructs a MD_SemanticCacheMgr object	
mvSemanticInit	construct and load (1) a cache for all Resolve stats,(2) a cache of Resolve stats mapping list, and (3) a cache of all NMS stats for one NMS_type and one techtype	TRUE or FALSE

Table 3-17 MD_SemanticCacheMgr Methods

3.3.7.3.2 Class MD_MvStatMapperMgr

Class name: MD_MvStatMapperMgr

Superclass: none

Class role: This class is responsible for providing a MD_MvStatMapper object for the next-to-be-processed Resolve stats.

Methods:

Name	Description	Return value
constructor	constructs a MD_MvResStatMgr object	
getNextResStatMapper	Retrieves the MD_MvResStatMapper object for the next-to-be-mapped Resolve stats.	TRUE or FALSE
getNumResStatMappers	Returns the number of mapper objects (i.e., number of Resolve stats objects) for one NMS_type and one techtype	TRUE or FALSE
reset	initialize a position token pointing to the begining of the list of the to-be-mapped Resolve stats for a dbload record.	
getColumnOrder	Returns the column position of the next-to-be-mapped Resolve stats	an integer

Table 3-18 MD_MvStatMapperMgr

3.3.7.3.3 Class MD_MvStatMapper

Class name: MD_MvStatMapper

Superclass: none

Class role: This class is responsible for providing a MD_MvStatMapping object for the next-to-be-processed Resolve stats.

Methods:

Name	Description	Return value
constructor	constructs a MD_MvResStatMapper object	
getNextResStatMapping	Retrieves the MD_MvResStatMapping object for the next-to-be-mapped Resolve stats.	TRUE or FALSE
reset	Reset the counter to point to the first mapping object in the mapping list	
getNumResStatMappings	Returns the number of mapping objects for one Resolve stats	TRUE or FALSE

Table 3-19 MD_MvStatMapper

3.3.7.3.4 Class MD_MvMapping

Class name: MD_MvMapping

Superclass: none

Class role: This class is responsible for providing all the mapping stats for the next-to-be-processed Resolve stats.

Methods:

Name	Description	Return value
constructor	constructs a MD_MvMapping object	
getNMSStat1	Retrieves the de_type of NMSStat1 within the mapping	A short integer
getNMSStat2	Retrieves the de_type of NMSStat2 within the mapping	A short integer
getNMSStat3	Retrieves the de_type of NMSStat3 within the mapping	A short integer
getRcaname1	Retrieves the rcaname of NMSStat1 within the mapping	A RWCString
getRcaname2	Retrieves the rcaname of NMSStat2 within the mapping	A RWCString
getRcaname3	Retrieves the rcaname of NMSStat3 within the mapping	A RWCString
getOperation1	Retrieves the operation following NMSStat1 within the mapping	A RWCString
getOperation2	Retrieves the operation following NMSStat2 within the mapping	A RWCString

Table 3-20 MD_MvMapping

4 INTERACTION DIAGRAMS

4.1 Description

This section provides several key scenario diagrams that will help understanding the working mechanism of the Metadata Engine.

4.2 Cache Creation and Configuration

This sub-section describes the steps to create and configure a Resolve stats cache.

1. The SemanticCacheMgr object creates the Resolve stats cache
2. The SemanticCacheMgr object starts to configure the cache
3. The cache tries to get configuration parmaters from the configuration file pointed by CK_Config object
4. The cache verifies and saves the configuration parmaters

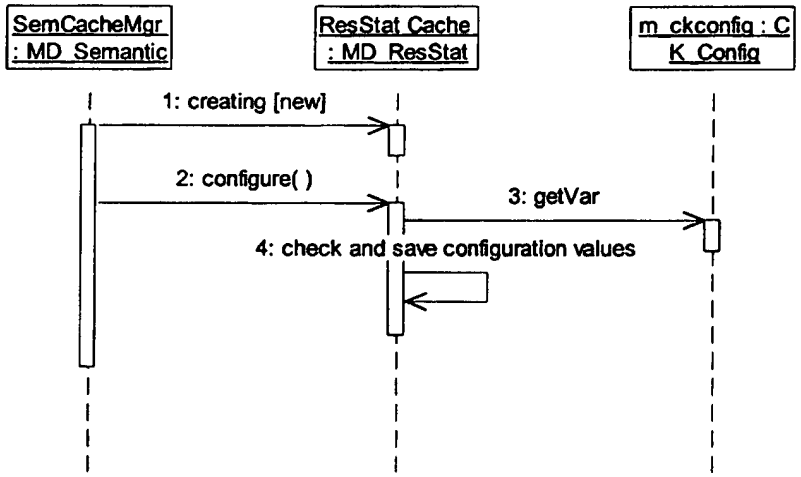


Figure 4-1 Resolve Statistics Cache - Creation and Configuration

4.3 Cache Loading

This sub-section describes the steps to load a Resolve stats cache which has been correctly created and configured.

1. The semanticCacheMgr invokes its load method to start load the cache
2. The cache invokes its private method m_load
3. The cache verifies if it is properly configured. It returns FALSE if not.

4.4 SyntacticCachrMgr Interacts with Syntactic Translator

This sub-section describes the steps by which the Syntactic Translaor (i.e., the Statistics Importer) interacts with the SyntacticCacheMgr to process each NMS statisitcs.

1. The SI creates a SyntacticCacheMgr object
2. The SI invokes the mvSyntacticInit method of SyntacticCacheMgr to create, configure and then load a cache of NMS stats for NMS_type
3. The SI invokes the mvRegister method of SyntacticCacheMgr to create, configure and then load a cache of NMS stats mappings for NMS_type and techtype. This step will be executed several times for all techtypes supported by the NMS_type
4. The SI invokes the mvRegisterSupplementalStat method of SyntacticCacheMgr to create, configure and then load (1) a cache of supplemental NMS stats for NMS_type, and (2) a cache for all entity whose stats all have been selected as being supplemental
5. The SI retrieves next-to-be-processed NMS stats and its associated entity from the NMS stats data file
6. The SI invokes the isNMSStatRequired method of SyntacticCacheMgr to check if the NMS stats is a core or extended stats. If the return code of the method is greater than 0, SI process continues to next step. If not, go to step 8
7. Output the NMS stats to the proper ROF file and then go back to step 5 until all NMS stats have been processed
8. The SI invokes the isASupplementStat method of SyntacticCacheMgr to check if the NMS stats is a supplemental stats. If the return code of the method is greater than 0, SI process continues to next step. If not, go back to step 5
9. Output the NMS stats to the supplemental ROF file and then go back to step 5 until all NMS stats have been processed

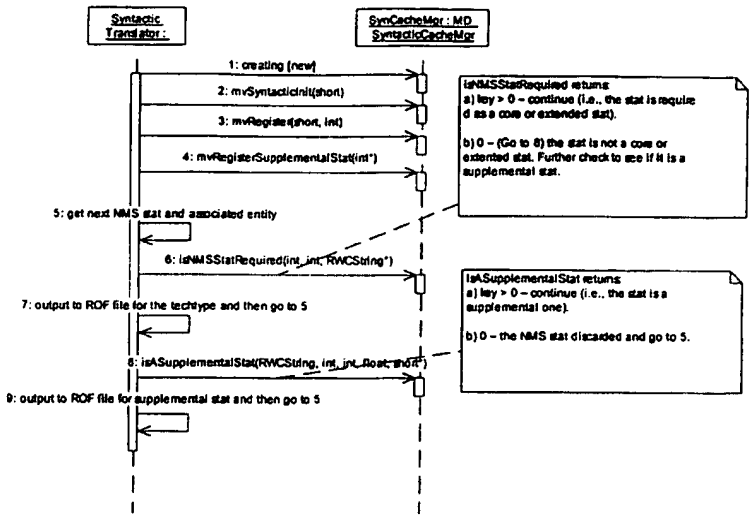


Figure 4-3 SyntactiCacheMgr interacts with Syntactic Translator

4.5 Semantic Translator interacts with Semantic Translation Utility Objects

This sub-section describes the steps by which the Semantic Translator (i.e., the MV Parser) interacts with the various Semantic Utility objects to process each NMS statisitcs from the ROF file.

1. The Parser creates a SemanticCacheMgr object
2. The Parser invokes the mvSemanticInit method of SemanticCacheMgr to create, configure and then load several caches (for NMS_type and the techtype) necessary for the semantic stage of stats translation
3. The Parser creates a ResStatMapperMgr object
4. The Parser invokes the reset method of the ResStatMapperMgr object to initialize the position counter of the object
5. The Parser invokes the getNextResStatMapper method of ResStatMapperMgr to get the mapper object for mapping several NMS statisitcs to one Resolve stats
6. The ResStatMapperMgr creates a ResStatMapper object
7. The ResStatMapperMgr returns to the Parser a pointer to the ResStatmapper objeect
8. The Parser invokes the getNumMappings method of the ResStatMapper object to retrieve the number of mappings for the Resolve statisitcs
9. The Parser invokes the getNextMapping method of the ResStatMapper object to retrieve the next mapping for the Resolve statisitcs
10. The MapperMgr creates a MvMapping object
11. The MapperMgr returns the pointer to the MvMapping object back to The Semantic Translator

12. The Parser invokes several methods of the mapping object to retrieve all the necessary information (e.g., the rcanames for all the involved NMS stats)
13. The Parser does something for those retrived information (e.g., stores them into its own memory storage) and then start processing every RCO objects in the ROF file until the end of the ROF file is reached

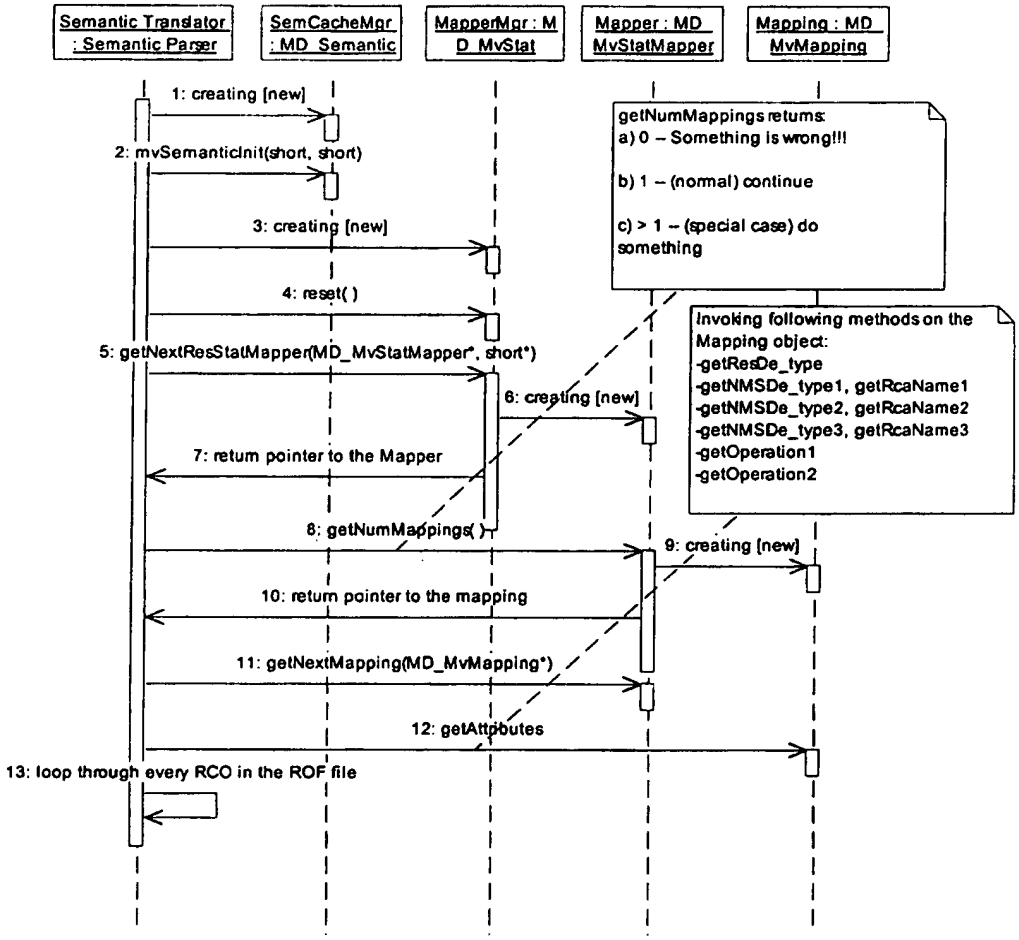


Figure 4-4 Semantic Translator interacts with Semantic Translation Utility Objects

Claims:

1. A method of converting data in one form to another, wherein the conversion is performed with metadata describing the original and final data.